Satisfiability Modulo Graph Theory for Task Mapping and Scheduling on Multiprocessor Systems

Weichen Liu, *Student Member*, *IEEE*, Zonghua Gu, *Member*, *IEEE*, Jiang Xu, *Member*, *IEEE*, Xiaowen Wu, *Student Member*, *IEEE*, and Yaoyao Ye, *Student Member*, *IEEE*

Abstract—Task graph scheduling on multiprocessor systems is a representative multiprocessor scheduling problem. A solution to this problem consists of the mapping of tasks to processors and the scheduling of tasks on each processor. Optimal solution can be obtained by exploring the entire design space of all possible mapping and scheduling choices. Since the problem is NP-hard, scalability becomes the main concern in solving the problem optimally. In this paper, a SAT-based optimization framework is proposed to address this problem, in which SAT solver is enhanced by integrating with a scheduling analysis tool in a branch and bound manner to prune the solution space efficiently. Performance evaluation results show that our technique has average performance improvement in more than an order of magnitude compared to state-of-the-art techniques. We further build a cycle-accurate network-on-chip simulator based on SystemC to verify the effectiveness of the proposed technique on realistic multiprocessor systems.

Index Terms—Multiprocessor, scheduling, design space exploration, satisfiability.

1 Introduction

Multiprocessor systems are becoming ubiquitous in today's embedded systems design. Exploiting concurrency inside applications is the key to take advantage of the parallel computation power of such systems. Thus, task mapping and scheduling is one of the most important issues in embedded multiprocessor systems design, where applications are often described by task graphs in high-level abstraction. If task workload and communication volume can be estimated or bounded in advance, static analysis can be applied at compile time, and the benefit and potential risk can be predicted at early design stage. Moreover, it is possible to apply exact analysis techniques that require long execution time but provide results with high quality. This paper discusses the possibilities of solving the difficult optimization problem of task mapping and scheduling optimally.

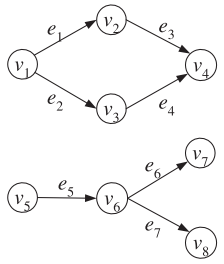
There are mainly two static scheduling policies for task graphs: time-based scheduling, which determines the precise time instant of each task execution; order-based scheduling, which invokes tasks in a predefined order. A special class of order-based scheduling is as-soon-as-possible (ASAP) scheduling, in which the order is determined naturally by the precedence constraints among tasks, i.e., a task becomes available for execution as soon as all its predecessors finish. Order-based scheduling can achieve higher processor utilization than time-based scheduling, especially if actor execution times show a large degree of variation. ASAP scheduling is optimal if hardware resource is unlimited, which does not hold true when resource constraint is imposed, e.g., limited number of processors. Thus, finding appropriate task execution order on processors is essential for scheduling of multiprocessor systems. Since the Boolean Satisfiability (SAT) techniques developed rapidly in the last two decades, and modern conflict-driven SAT solvers based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [1], [2] are very powerful on solving large problems, it motivates us to apply SAT techniques to this optimization problem [3]. The main issue is how to manage the optimization space efficiently.

In this paper, we investigate the implementation detail of DPLL in SAT solver, and embed a graph theory solver specific to our problem inside the DPLL search procedure to accelerate the exploration to the exponential design space. The objective is to discover how to prune the search space effectively to enable the solver to find optimal solutions for a larger class of problems. Our main contribution is the proposal of a formal optimization framework with effective constraint learning and search space pruning, realized by the tight integration of the SAT solver with graph theory solver. We also develop a cycle-accurate simulator for network-on-chip systems using SystemC and verify the effectiveness of our technique by running simulations on realistic applications.

The rest of the paper is structured as follows: we formulate the problem and discuss related works in Section 2, we explain the analysis techniques for mapping and scheduling task graphs in Section 3, present the SAT-based optimization framework in Section 4; performance evaluation results are given in Section 5 and we conclude the paper in Section 6.

- *W. Liu, J. Xu, X. Wu, and Y. Ye are with the Hong Kong University of Science and Technology, Hong Kong, China. E-mail: {weichen, eexu, wxxaf, yeyaoyao}@ust.hk.*
- *Z. Gu is with the Zhejiang University, Hangzhou, China. E-mail: zgu@zju.edu.cn.*

Manuscript received 2 Dec. 2009; revised 3 Aug. 2010; accepted 15 Sept. 2010; published online 9 Nov. 2010.
Recommended for acceptance by A. Zomaya.
For information on obtaining reprints of this article, please send e-mail to: tpds@computer.org, and reference IEEECS Log Number TPDS-2009-12-0596.
Digital Object Identifier no. 10.1109/TPDS.2010.204.



	WCET (s)
v_1	3
v_2	3
v_3	3
v_4	7
v_5	2
v_6	3
v_7	5
v_8	6

	WCTT (s)
e_1	9
e_2	5
e_3	7
e_4	4
e_5	3
e_6	10
e_7	4

Fig. 1. An example task graph with data of worst-case execution times and communication delays (source: [3]).

2 Problem Formulation and Related Work

A task graph $G(V, E)$ is a directed acyclic graph (DAG), where $V = \{v_1, \ldots, v_n\}$ is the set of computation tasks, and $E \subseteq V \times V$ is the set of data dependency constraints between tasks. A homogeneous multiprocessor platform $P = \{p_1, \ldots, p_m\}$ consists of m identical processors connected by a communication network. Each processor contains dedicated communication component so that computation and communication can be executed in parallel. Tasks are executed sequentially without preemption on a processor. Every task is able to run on any processor without difference. Worst-case execution time (WCET) of task $v \in V$ is given by $t(v)$, and communication volume on edge $(v_i, v_j) \in E$ is given by $c(v_i, v_j)$. If two tasks with data dependence are mapped to the same processor, intertask communication is implemented by data sharing in local memory, and no communication latency is incurred. If two tasks are mapped to different processors, communication between them is implemented by data transfer through network links, and communication latency corresponds to the worst-case network transmission time (WCTT). In this work, we assume the WCET values of tasks and WCTT values of intertask communications are known, and focus on optimizing the mapping and scheduling of a task graph under these assumptions. An example task graph is shown in Fig. 1.

Given a task graph $G(V, E)$ and a homogeneous multiprocessor platform P, the design problem is to find a *mapping* $M : V \rightarrow P$ for each task in V to a processor in P and a *schedule* $S : V \rightarrow N$ for the tasks assigned to the same processor, where each task in V mapped to a processor in P is assigned a natural number indicating its unique sequence number for execution on P, so that all data dependency constraints in E are satisfied, and the makespan $m(G)$ is minimized. *Makespan*, also called schedule length, is defined as the time difference between the start of the earliest task and the finish of the latest one. Suppose in a mapping and scheduling result, the start time of each task $v \in V$ is $s(v)$. Then, $m(G) = \max_{v \in V}[s(v) + t(v)] - \min_{v \in V} s(v)$.

The problem to find optimal mapping and scheduling solution is NP-hard [4]. Various heuristic algorithms, branch and bound methods, evolutionary algorithms and constraint formulations have been devised to solve it. Kwok and Ahmad [5] compare the performance of different approaches. In order to obtain provably optimal result in the entire search space, a Mixed-Integer Linear Programming (MILP)-based approach [6] and a Logic-Based Benders Decomposition (LBBD)-[7] based approach [3] are evaluated as efficient techniques. In [6], the problem is formulated by a variety of MILP formulations, and solved by commercial MILP solver ILOG CPLEX. In [3], by applying decomposition techniques, the problem is divided into a master problem of finding a feasible mapping and scheduling solution, formulated by a group of boolean constraints and solved by a SAT solver, and a subproblem of evaluating the solution by graph theory and providing additional boolean constraint feedback to the master problem to prune the search space. Compared to the MILP-based approach, the LBBD-based approach iteratively solves simpler versions of the same problem and learns constraints at each iteration, which is proved to be more efficient in solving complex problems [3]. The technique proposed in this paper is an improvement to the LBBD-based approach, which does not need the master problem to be solved completely before moving to the subproblem, but with branch and bound, a partial solution to the master problem can help cut the solution space more efficiently. Heuristic algorithms based on list scheduling [8], [9] and stochastic techniques like evolutionary algorithms [5] can also be applied to this problem. Compared to formal methods, the main drawback of these techniques is on the quality of the obtained solution [3].

Metzner and Herde [10] presented a SAT-based approach to address the problem of periodic task allocation for distributed real-time systems, in which SAT solver is enhanced with real-time scheduling theory. The author typically addresses schedulability analysis of priority-driven preemptive real-time scheduling algorithms, but not static scheduling of task graphs. Liu et al. [11] propose a SAT-based technique to optimize throughput of homogeneous synchronous dataflow graphs on multiprocessor platforms, in which maximum cycle ratio analysis is integrated with SAT solver in two manners. In this paper, we apply similar ideas with the integrated approach in [11], while target the problem of makespan optimization of task graph scheduling on multiprocessors. A more closely integrated optimization framework is constructed that replaces the binary search solution in [10] with relevant backjumping and continued search operations that accelerates the design space exploration.

3 Task Mapping and Scheduling Analysis

In this section, we introduce a *Makespan Analysis Engine (MAE)*, which is used for analyzing and evaluating a mapping and scheduling solution for a task graph. In the context of this paper, we use *decision* to denote the choice of a single task mapping or scheduling, and use *solution* to denote the decisions on a set of task mapping and scheduling. A solution can be either full or partial. A *full solution* indicates all the decisions of task mapping and scheduling, while a *partial solution* includes only part of the whole decisions, and some task mapping or scheduling decision is still unknown. The MAE is used for evaluation of a design point in the design space, which is a task mapping and scheduling solution, and provides useful information for the design space exploration framework. It has two functions: calculate the *makespan* of a task graph with a given solution, and analyze the *reason* that the current task mapping and scheduling solution constrains the makespan.

A *reason* for a mapping and scheduling solution is a part of the solution that determines the makespan directly. It is the critical part of the solution that restricts the makespan to become smaller. We use a simple example to explain what a

reason is. Suppose tasks v_1, v_2 in a task graph with WCETs of 100 and 200 time units are mapped to the same processor. Then, a simple fact is makespan of any solution containing this decision can never be smaller than 300. "Tasks v_1 and v_2 on the same processor" is a *reason* that makespan is constrained to be no less than 300, which means all the scheduling decisions including "v_1, v_2 on the same processor" cannot lead to a better solution than 300. It holds true no matter how the other tasks of the task graph are mapped and scheduled, since the reason part always constrains the makespan. Therefore, we should avoid the recurrence of the reason in future search in order to obtain better makespan, i.e., never map v_1 and v_2 on the same processor. This information helps prune the search space of possible mapping and scheduling decisions. A reason can be extracted from a single mapping and scheduling solution, while it is shared by a class of mapping and scheduling solutions with the same condition. A reason is more powerful in pruning search space if it contains less mapping and scheduling decisions, since it covers the huge design space of all possible choices on the decision of uninvolved items.

The input to MAE is a task graph and a mapping and scheduling solution, and the output is the analysis result of makespan and reason. As shown in Algorithm 1, MAE includes three functional modules. The first module transforms the task graph into a new directed graph (so called platform specific model, PSM) according to the mapping and scheduling decisions, using the technique described in Section 3.1. The new graph contains additional constraints modeling the mapping and scheduling decisions. The second module applies cycle detection and makespan calculation to the PSM. If a cycle is detected, each task on the cycle causally depends on its predecessor, either by data dependency constraints or by static order constraints. It causes a deadlock, and the mapping and scheduling solution is infeasible. The makespan in this case can be viewed as infinity. Otherwise, if the PSM is acyclic, we apply critical path analysis to the PSM to obtain the value of makespan. Details are defined in Section 3.2. The third module analyzes the reason that forms either the deadlock cycle or the critical path, which is a subset of the mapping and scheduling decisions in the solution.

Algorithm 1. Makespan Analysis Engine (MAE)
Require: a task graph G, a mapping and scheduling solution D
1: $makespan = \infty$, $reason = \emptyset$
2: $psm =$ ConstructPSM(G, D)
3: **if** DetectCycle($psm, cycle$) returns $true$ **then**
4: $reason =$ GenerateReason($psm, G, cycle$)
5: **else**
6: $makespan =$ GetCriticalPath($psm, path$)
7: $reason =$ GenerateReason($psm, G, path$)
8: **end if**
9: **return** $makespan$, $reason$

3.1 PIM and PSM

We call the input task graph "platform independent model (PIM)," which contains only application logic. Given a mapping and scheduling solution, we can transform a PIM to a "platform specific model (PSM)" [12]. The aim of the transformation is to model platform constraints, including network delay and processor sharing, by adding new elements to the graph, so that different mapping and scheduling decisions can be easily addressed by the PIM-to-PSM transformation, and makespan calculation is simplified to a uniform method that applies on different PSMs. The rules to generate PSM from PIM is two parts. First, assign communication latency to each edge according to the mapping decisions on the two end tasks: $c(v_i, v_j)$ if v_i and v_j are mapped to different processors; zero if v_i and v_j are mapped to the same processor or not both mapped yet. Second, add a new directed edge (v_i, v_j) for ordering constraint if the scheduling decision v_i precedes v_j is made. A PSM contains not only all the information in the original PIM, but also task mapping and scheduling constraints, by means of enabling edge communication delays and adding new directed edges.

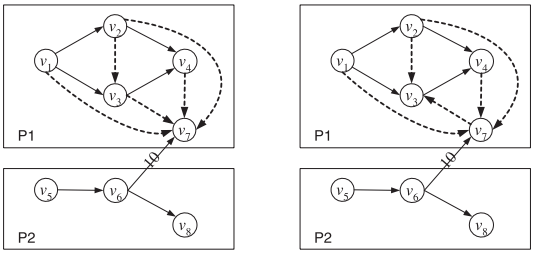


Fig. 2. Example PSMs, (a) Critical path $v_1v_2v_3v_4v_7$, (b) Deadlock cycle $v_3v_4v_7v_3$.

Suppose the example given in Fig. 1 is mapped and scheduled on a two-processor platform, and we have an example solution like this: tasks v_1, v_2, v_3, v_4, v_7 are mapped to the first processor and v_5, v_6, v_8 to the second one; the static order schedules on the two processors are $v_1v_2v_3v_4v_7$ and $v_5v_6v_8$, respectively. By applying the transformation rules, we obtain the PSM as shown in Fig. 2a. The processor panels (P1 and P2) are added for clarity. Edge (v_6, v_7) is marked with weight 10, which means the interprocessor communication delay between tasks v_6 and v_7 is 10 time units. All the other edges in this example are internal edges within a processor. The weights on these edges are zero, omitted for clarity in the figure. The dotted edges are new edges depicting scheduling constraints, e.g., edge (v_2, v_3) constrains v_3 must start execution after v_2 finishes. Edges (v_1, v_7), (v_2, v_7), (v_3, v_7), and (v_4, v_7) are inserted for the same reason. These order constraints combine to represent the schedule on the processor. Note that the ordering constraints between tasks with transitive closure relations can be omitted, e.g., edge (v_1, v_4) and (v_5, v_8) are not needed in the PSM. If in another solution, the schedule on P1 is changed to $v_1v_2v_4v_7v_3$ while the other decisions remain the same, the obtained PSM is shown in Fig. 2b, where there is a deadlock cycle $v_3v_4v_7v_3$.

3.2 Makespan Analysis

Makespan analysis is a group of operations applied to the PSM, including deadlock detection, makespan calculation and reason extraction. There are two situations: if a cycle exists in the PSM, which indicates cyclic dependencies between tasks (deadlock) exist, the task graph is not schedulable in this case, and the corresponding mapping and scheduling solution is an invalid one; if no cycle exists, the task graph is schedulable, and the makespan is

determined by the path with the largest delay in the PSM (critical path). The delay of a path is the sum of task execution times and edge delays on it.

In order to achieve effective feedback during design space exploration, we need to analyze the reason for a mapping and scheduling solution. We compare the PSM with PIM, and extract the platform specific information (mapping and scheduling decisions) on the deadlock cycle or the critical path. There are two situations when traversing the edges in the PSM. If an edge exists in PSM but not in PIM, it is an edge representing ordering constraint between its end tasks. If an edge exists in both PSM and PIM, the weight of the edge (communication latency) represents mapping constraint on the two end tasks: the latency is zero if the tasks are mapped to the same processor, or a positive value if to different processors. The reason is formed by combining these mapping and scheduling decisions together. It is always a subset of the mapping and scheduling solution used for constructing the PSM. In the deadlock case, a reason only contains ordering information and ignores mapping, since whether communication delay is zero or positive, decided by mapping, does not impact deadlock. In fact, any cycle in the PSM is the reason for deadlock, and any path with delay longer than a threshold is the reason for makespan constraining. However, detecting all cycles in a graph or finding all paths with delay longer than a threshold is extremely time consuming. Therefore, we only collect one deadlock cycle or one critical path.

Applying makespan analysis to the example PSM given in Fig. 2a, we obtain the critical path $v_1v_2v_3v_4v_7$, and the makespan is 21 time units. By reason extraction, we know the combination of ordering constraints (v_2, v_3) and (v_4, v_7) leads to the critical path, and thus is the reason. For the PSM in Fig. 2b, a deadlock cycle $v_3v_4v_7v_3$ is detected. The combination of ordering constraints (v_4, v_7) and (v_7, v_3) is extracted from the cycle as the reason.

Consider the relationship between a partial solution and any full solution obtained by extending the partial solution. According to the rules of PSM construction, having more mapping and scheduling decisions adds more constraints to the generated PSM, and thus can never decrease the value of the makespan. It means, *makespan of a partial solution forms a lower bound on the makespan of any full solution obtained by extending the partial solution*. This conclusion is useful for building the branch and bound search framework, as proposed in Section 4. When exploring the design space, we can keep the best solution obtained so far as a guard, and evaluate a mapping and scheduling result earlier: as long as makespan of a partial solution is worse than the guard already, no further extension to a full solution is needed, since no better solution can be found. The formal proof of the observation is given in the supplementary file, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TPDS.2010.204.

4 SAT-Based Optimization Framework

The entire problem is a constraint optimization problem: a feasible solution to the problem satisfies all the data dependency constraints defined in a task graph (hard constraints), whereas an optimal solution has minimized makespan among all feasible solutions (soft constraints). There are two key steps to solve the problem: traverse the design space to generate candidate solutions; evaluate candidate solutions and provide instructional feedback to the following design space exploration. We use SAT to build the optimization framework to accomplish the first problem, and use MAE proposed in Section 3 to solve the second problem of analysis and feedback. We first discuss the search framework, and propose the SAT encoding of the problem in Section 4.2.

4.1 Framework Description

According to the lower bound rule proposed at the end of Section 3.2, our problem can be solved by branch and bound algorithms: storing the makespan of the best solution found during search and use it for pruning the search space. More precisely, whenever the algorithm encounters a partial solution that cannot be extended to form a solution with better makespan than the stored best makespan, the algorithm backtracks in the search tree, instead of trying to extend this solution. Specifically, a SAT instance, formulating all data dependency constraints, is generated as the input to the SAT solver. As the SAT solver runs, partial (or full) assignments to the boolean variables are extracted and sent to the MAE for evaluation, and the best solution is maintained during the progress. MAE works as the subproblem solver to calculate the makespan and analyze the reason corresponding to the variable assignments, as stated in Section 3. Afterward, if branch-pruning condition is satisfied, or a feasible solution is found, a blocking clause is created to prevent the recurrence of the reason, and added to the SAT instance as additional constraint to restrict future search. When the SAT solver cannot find any more feasible solution, the entire solution space has been covered and the solution maintained with best makespan is the optimal solution. The overall algorithm is described in Algorithm 2. The algorithm is built upon the structure of DPLL algorithms, and the subproblem solver MAE is integrated with branch and bound.

Algorithm 2. SAT-based optimization framework
Require: a task graph G, a multiprocessor platform P

```
1: bestSol = ∅
2: GenerateConstraints(G, P)
3: while true do
4:    PropagateAssignment()
5:    if DetectConflictBySAT() returns true then
6:       if TopLevelConflict() then
7:          break
8:       else
9:          LearnConstraintBySAT()
10:         NonChronologicalBacktrack()
11:      end if
12:   else if DetectConflictByMAE() returns true then
13:      LearnConstraintByMAE()
14:      NonChronologicalBacktrack()
15: else if FullAssignment() then
16:      LearnConstraintByMAE()
17:      bestSol = UpdateBestSolution()
18:      NonChronologicalBacktrack()
19:   else
20:      DecideAssignment()
21:   end if
22: end while
23: return bestSol and its makespan
```

The input to Algorithm 2 is a task graph G and a multiprocessor platform P, and the output is the optimal mapping and scheduling result $bestSol$ with minimum makespan. First, a SAT instance is generated as the input to the SAT solver (Line 2). Possible task mapping and scheduling choices under data dependency constraints in G are formulated in the SAT instance, as described in Section 4.2. The entire design space exploration is proceeded in the while loop (Line 3-22), in the form of a binary search tree of the boolean variables, maintained by the SAT solver. The main operations are decision (Line 20), propagation (Line 4), and backtracking (Line 10, 14, 18), which drive the search procedure of SAT solving. Detailed working strategies of modern SAT solvers can be found in [13]. Some basic operations are described as follows: $DetectConflictBySAT()$ (Line 5) is defined by the SAT solver to check if some constraint is violated by the current assignment. If the conflict happens at the top level of the search tree, indicating no assumption to any boolean variable assignment is made, the SAT problem is unsatisfiable (Line 6-7). Otherwise, the SAT solver analyzes the violated constraint and generates a new clause to indicate the reason that causes the conflict (Line 9). The new clause is added to the SAT instance to instruct the following search. This procedure is called *conflict-driven clause learning*. If all variables are assigned without conflict (Line 15), a satisfying solution to the problem is found. Normally, the SAT solver should stop working and output the feasible boolean assignments. However, we make modifications here to make it become an optimization framework which iteratively finds better solutions until the entire search space is covered.

Compared to the LBBD solution in [3], we enhance the SAT solver with two modifications by integrating the MAE to it. First, when a satisfying solution is found (Line 15), instead of ending the search, we translate the result into a task mapping and scheduling solution, and pass it to the MAE for makespan analysis. MAE will return the makespan result, together with a new boolean clause depicting the *reason* (Line 16). Then, the best solution is updated (Line 17) and the SAT solver plays backtracking to a state where not all the decisions of the reason hold (Line 18). As the SAT solver runs, it will lead to a different solution for the learned constraint has abandoned search from the solution space covered by the reason. Second, when no conflict is detected by SAT for a partial solution, instead of making new decisions immediately (Line 20), makespan analysis on the current assignment is performed (Line 12-14). Similarly, the partial assignment is translated into a partial task mapping and scheduling solution and passed to MAE. The function $DetectConflictByMAE()$ (Line 12) returns $true$ when the makespan result is worse than the best makespan maintained. We treat it as "constraint violation." Although no conflict happens to the SAT clauses, the current partial assignment cannot possibly lead to a better solution, according to the lower bound rule. Thus, the reason is translated into a boolean clause and added to the SAT instance (Line 13). Backtracking is then performed accordingly (Line 14).

Benefiting from the enhancement, design space exploration of the optimization problem is achieved by running the constraint satisfiability solver only once. The search tree branch pruning strategy of the SAT solver is effectively enhanced by communicating with the additional component MAE actively with branch and bound. The realistic meaning of the boolean variables is utilized by MAE, who provides useful feedback to the SAT solver from the knowledge of this specific problem beyond SAT solving itself. In our problem, the boolean variables representing mapping and scheduling decisions are used to generate additional constraints by analyzing reasons using graph theory. Some useful heuristics embedded in the optimization framework for efficiency improvement can be found in the supplementary file.

TABLE 1
Definition of Ordering Variables between Tasks

		$x_d(v_i, v_j)$	$x_d(v_j, v_i)$
v_i, v_j on two processors		0	0
v_i, v_j on one processor	v_i procedes v_j	1	0
	v_j procedes v_i	0	1
Never happen		1	1

4.2 SAT Encoding of the Problem

The problem formulation is given in Section 2. We define three types of boolean variables for SAT encoding of the problem: mapping variables x_a indicate task mapping to processors, communication variables x_c indicate interprocessor communications between tasks, and ordering variables x_d indicate the execution orders of tasks mapped to the same processors. The accurate definitions are described by the following equations [3]. Note we define two ordering variables $x_d(v_i, v_j)$ and $x_d(v_j, v_i)$ for each pair of tasks v_i and v_j to denote the three possible states: undefined order when they are mapped to different processors, and two possible orders v_iv_j or v_jv_i when they are mapped to the same processor, as described in Table 1. Since [3] only uses full solution for makespan analysis, there is no undetermined state for ordering variables. Let $G^T = (V, E^T)$ be the transitive closure of task graph $G = (V, E)$: $(v_i, v_j) \in E^T$ if there exists a directed path from v_i to v_j in G. In this case, task v_j must execute after v_i since each task on the path causally depends on its predecessor in sequence. Therefore, the execution order between v_i and v_j is fixed and variable x_d is not needed for modeling the order between v_i and v_j. From the definitions, the solution to the problem can be extracted from mapping variables x_a and ordering variables x_d. Communication variables x_c denote the existence of communication latency, which are especially introduced to improve the efficiency of information exchange between the SAT solver and the MAE: only x_c and x_d are used in the MAE for PIM to PSM transformation.

$$\forall v \in V, \forall p \in P, x_a(v,p) = \begin{cases} 1, & \text{if } v \text{ is mapped to } p \\ 0, & \text{otherwise.} \end{cases}$$

$$\forall (v_i, v_j) \in E, x_c(v_i, v_j) = \begin{cases} 1, & \text{if } v_i \ \& \ v_j \text{ are mapped to} \\ & \quad \text{different processors;} \\ 0, & \text{otherwise.} \end{cases}$$

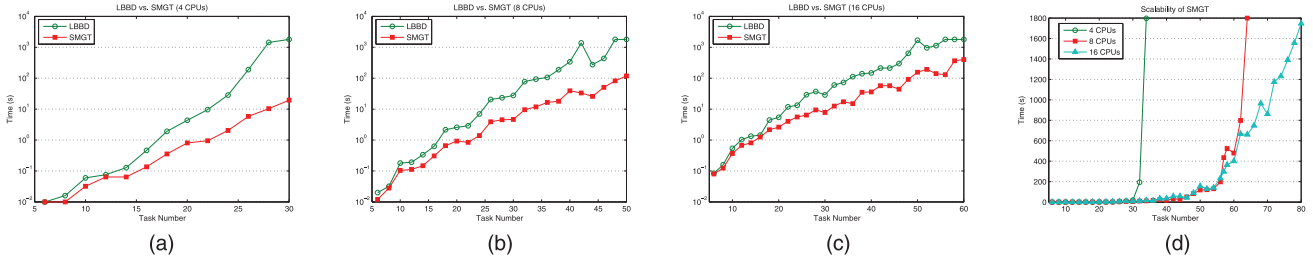


Fig. 3. Comparisons of the running times of LBBD and SMGT on three multiprocessor platforms. (a) 4 CPUs. (b) 8 CPUs. (c) 16 CPUs. (d) Scalability of SMGT.

$$\forall v_i, v_j \in V, (v_i, v_j) \notin E^T, (v_j, v_i) \notin E^T,$$
$$x_d(v_i, v_j) = \begin{cases} 1, & \text{if } v_i \text{ precedes } v_j; \\ 0, & \text{otherwise.} \end{cases}$$

The following boolean formulas are used to generate the initial SAT instance to the SAT solver. Similar constraints are first proposed in [3]. Intuitively, constraints A1 and A2 indicate each task is mapped to exactly one processor; constraints C1 and C2 indicate interprocessor communication is introduced only if two communicating tasks are mapped to different processors; constraints D1 indicate the execution order of two tasks when they are mapped to the same processor, and D2 indicates the two ordering variables should be both zero if the two tasks are mapped to different processors.

$$\text{(A1)}\ \forall v \in V, \bigvee_{p \in P} x_a(v, p).$$
$$\text{(A2)}\ \forall v \in V, \forall p_i, p_j \in P, p_i \neq p_j, x_a(v, p_i) \wedge x_a(v, p_j) \Rightarrow 0.$$
$$\text{(C1)}\ \forall (v_i, v_j) \in E, \forall p_k, p_l \in P, p_k \neq p_l,$$
$$x_a(v_i, p_k) \wedge x_a(v_j, p_l) \Rightarrow x_c(v_i, v_j).$$
$$\text{(C2)}\ \forall (v_i, v_j) \in E, \forall p \in P, x_a(v_i, p) \wedge x_a(v_j, p) \Rightarrow \neg x_c(v_i, v_j).$$
$$\text{(D1)}\ \forall v_i, v_j \in V, (v_i, v_j) \notin E^T, (v_j, v_i) \notin E^T, \forall p \in P,$$
$$x_a(v_i, p) \wedge x_a(v_j, p) \Rightarrow x_d(v_i, v_j) \vee x_d(v_j, v_i),$$
$$x_d(v_i, v_j) \wedge x_d(v_j, v_i) \Rightarrow 0.$$
$$\text{(D2)}\ \forall v_i, v_j \in V, (v_i, v_j) \notin E^T, (v_j, v_i) \notin E^T, \forall p_k, p_l \in P,$$
$$p_k \neq p_l, x_a(v_i, p_k) \wedge x_a(v_j, p_l) \Rightarrow \neg x_d(v_i, v_j) \wedge \neg x_d(v_j, v_i).$$

Let V_c and E_c be the set of vertices and edges on the deadlock cycle or critical path obtained by the MAE, respectively. The constructed boolean formula as a blocking clause for deadlock cycle is given in (1), and the one for critical path is given in (2). The intuitive meaning of the formulas is: as long as these platform elements (interprocessor communications and static order schedules) are granted simultaneously in future search, no matter how other design decisions are made, either the same deadlock cycle or the same critical path will appear again in the constructed PSM, and constrain a better solution to be derived. Therefore, we formulate a blocking clause to prevent their co-occurrence and add it to the SAT instance to restrict future search.

$$\bigwedge_{(v_k, v_l) \in E_c - E} x_d(v_k, v_l) \Rightarrow 0 \quad (1)$$

$$\left(\bigwedge_{\substack{v \in V_c - V, \\ (v_i, v), (v, v_j) \in E_c}} x_c(v_i, v_j) \right) \wedge \left(\bigwedge_{(v_k, v_l) \in E_c - E} x_d(v_k, v_l) \right) \Rightarrow 0 \quad (2)$$

5 Performance Evaluation

We conduct extensive simulations to evaluate the performance of the proposed approach. In [3], sufficient experimental results have been provided to show the advantages of formal methods on the provable optimality against heuristic and stochastic algorithms. Therefore, our focus is on the comparison among formal techniques. Algorithm efficiency in terms of running time complexity is the main concern.

The prototype of our approach is implemented by integrating the MAE component (written in C++) to an efficient open source SAT solver Minisat2 [13], the winner of SAT-Race 2008. Since MAE is based on graph theory, we name the overall implementation *SAT Modulo Graph Theory (SMGT)*, from the perspective of SAT Modulo Theories (SMT) [14]. We use *LBBD* to represent the LBBD-based technique proposed in [3], which has been evaluated to be more efficient than other formal techniques like MILP. Extensive experiments are performed to compare the efficiency of SMGT and LBBD, measured by the running times of the algorithms. All the experiments are conducted on a workstation running Fedora Linux Core with $4 \times$ AMD Opteron 844 (1.8 GHz) CPU and 8 GB RAM. We use a utility program *Memtime* to measure the running time of the algorithms.

We use two sets of benchmarks in the experiment. A set of random task graphs are used to evaluate the statistical performance of our technique, and a realistic application, H.264 HDTV decoder, is used to demonstrate the practicability and scalability of our technique to real-world large applications. Three multiprocessor hardware systems are used in the experiment, containing 4, 8, and 16 identical processors, respectively. The communication subsystem is implemented by the scalable NoC [15].

We use *TGFF* [16] to generate random task graphs with the following input parameters: WCET of each task ranges from 10 to 50 time units; WCTT of each intertask communication ranges from 5 to 30 time units; all other parameters are generated totally randomly. The size of a problem's design space is dependent on many problem-specific factors, including number of tasks and processors, shape and edge density of the graph, etc. The difficulty of a problem cannot be simply measured by the number of tasks. However, since the difficulty grows roughly exponentially with the number of tasks, we use task number as a rough gauge to catalog task graphs. We generate 10 task graphs for each catalog and always calculate their average as the result of that task number. Timeout is set to 30 minutes for all runs. Fig. 3 shows the comparisons of

TABLE 2
Performance Comparison on H.264 Decoder

No. CPUs	Makespan		Time (s)	
	LBBD	SMGT	LBBD	SMGT
4	559	513	$> 10^6$	23025.6
8	431	386	$> 10^6$	6527.4
16	380	379	$> 10^6$	55.3
32	379	379	1400.1	126.1
64	379	379	2406.4	465.2

running times of algorithms SMGT and LBBD on the three hardware platforms. The x-axis is marked by task numbers. The y-axis is the algorithm running times, drawn with a log scale since the difference in running times can be quite large for task graphs with different task numbers. Experimental results show that our approach consistently outperforms the LBBD-based approach in more than an order of magnitude (on average 12x speedup for the test cases). The advantage of our algorithm benefits from the more effective search tree pruning strategy with branch and bound by invocation of MAE on partial solutions, while the LBBD-based approach only invokes the analysis engine on full solutions after all mapping and scheduling decisions are made. Not only the backtracking decision is made earlier, but also the learned constraint is stronger in pruning search space. Fig. 3d shows the scalability of our algorithm with the 30 minutes timeout threshold. Within the time limit, task graphs with around 35, 65, and 80 tasks can be solved optimally on the three hardware platforms, respectively. Memory requirement is not growing fast as problem size increases. Our algorithm performs more robustly than related state-of-the-art techniques. More performance analysis on metrics in SAT solvers can be found in the supplementary file.

We conduct a case study on H.264 HDTV decoder to exploit the practicability and scalability of our technique to realistic applications. H.264 is an ITU-T recommended multimedia standard with improved coding efficiency and network-friendly design [17]. It has been adopted by new storage standards, such as Blue-ray and HD DVD, and is a promising candidate for HDTV broadcast. Nevertheless, H.264 is more complex than previous video standards like MPEG2, and precise scheduling is more helpful on performance improvement for MPSoC design of high resolution H.264 systems (e.g., 1080p). In [18], the functional parallelization of H.264 decoding is parsed into macro blocks (MB). We derive a prototype task graph with 51 task nodes from H.264 decoder. WCETs of tasks and WCTTs of edge communications are obtained by profiling. We compare the performance of SMGT and LBBD on several hardware platforms. As shown in Table 2, optimal results are obtained by SMGT for all the cases within limited time, while LBBD-based approach does not finish within 20 hours for the first three cases, and the makespan of the best solutions found is on average 10 percent larger than the optimal ones by SMGT. The results shows that our technique is more applicable to real-world applications due to better scalability. In addition, the optimal makespan is reducing with the number of available processors increasing (from 513 on 4 CPUs to 386 on 8 CPUs, and 379 on 16 CPUs), while the benefit of increasing processors is less obvious when the number grows to certain degree. For example, the optimal makespan of H.264 decoder is reduced by 24.7 percent when increasing the processor number from four to eight, while this rate becomes 2.1 percent when increasing it from 8 to 16. Further information can be found in the supplementary file.

6 Conclusion

In this paper, we present a formal technique to explore the design space of mapping and scheduling task graphs on multiprocessor systems. State-of-the-art techniques, such as MILP and LBBD-based approaches, are limited from achieving optimality by prohibitively long running times. To this end, we have proposed a SAT-based optimization framework with greatly improved efficiency. SAT solver is enhanced with a dedicated build-in component for task mapping and scheduling analysis, and effective solution space pruning is achieved by this integration in the branch and bound manner. Provably optimal solution is guaranteed to be found by our technique, and the scalability in terms of algorithm running time is improved in more than an order of magnitude, compared to state-of-the-art techniques.

Acknowledgments

This work was supported by RGC GRF 621108, Hong Kong.

References

[1] M. Davis and H. Putnam, "A Computing Procedure for Quantification Theory," *J. ACM*, vol. 7, no. 3, pp. 201-215, 1960.
[2] M. Davis, G. Logemann, and D. Loveland, "A Machine Program for Theorem-Proving," *J. Comm. ACM*, vol. 5, no. 7, pp. 394-397, 1962.
[3] N. Satish, K. Ravindran, and K. Keutzer, "A Decomposition-Based Constraint Optimization Approach for Statically Scheduling Task Graphs with Communication Delays to Multiprocessors," *Proc. Conf. Design, Automation and Test in Europe (DATE '07)*, pp. 57-62, 2007.
[4] Y. Kwok and I. Ahmad, "Benchmarking the Task Graph Scheduling Algorithms," *Proc. Int'l Parallel Processing Symp.*, pp. 0531-0537, 1998.
[5] Y.-K. Kwok and I. Ahmad, "Static Scheduling Algorithms for Allocating Directed Task Graphs to Multiprocessors," *ACM Computer Survey*, vol. 31, no. 4, pp. 406-471, 1999.
[6] K. Keutzer, K. Ravindran, N. Satish, and Y. Jin, "An Automated Exploration Framework for Fpga-Based Soft Multiprocessor Systems," *Proc. Hardware/Software Codesign and System Synthesis, CODES+ISSS '05, Third IEEE/ACM/IFIP Int'l Conf.*, pp. 273-278, Sept. 2005.
[7] J.N. Hooker, "Logic-Based Benders Decomposition," *Math. Programming*, vol. 96, pp. 33-60, 2003.
[8] G.C. Sih and E.A. Lee, "A Compile-Time Scheduling Heuristic for Interconnection-Constrained Heterogeneous Processor Architectures," *IEEE Trans. Parallel Distributed Systems*, vol. 4, no. 2, pp. 175-187, Feb. 1993.
[9] T. Davidović and T.G. Crainic, "Benchmark-Problem Instances for Static Scheduling of Task Graphs with Communication Delays on Homogeneous Multiprocessor Systems," *Computers Operations Research*, vol. 33, no. 8, pp. 2155-2177, 2006.
[10] A. Metzner and C. Herde, "Rtsat—An Optimal and Efficient Approach to the Task Allocation Problem in Distributed Architectures," *Proc. 27th IEEE Int'l Real-Time Systems Symp. (RTSS '06)*, pp. 147-158, 2006.
[11] W. Liu, M. Yuan, X. He, Z. Gu, and X. Liu, "Efficient Sat-Based Mapping and Scheduling of Homogeneous Synchronous Dataflow Graphs for Throughput Optimization," *Proc. Real-Time Systems Symp. (RTSS '08)*, pp. 492-504, 2008.

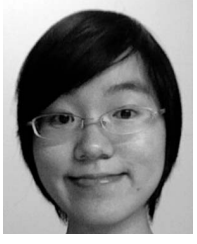
[12] D. Gracanin, S.A. Bohner, and M. Hinchey, "Towards a Model-Driven Architecture for Autonomic Systems," *Proc. IEEE Int'l Conf. Eng. of Computer-Based Systems*, p. 500, 2004.
[13] N. Eén and N. Sörensson, "An Extensible Sat-Solver," *SAT*, E. Giunchiglia and A. Tacchella, eds., pp. 502-518, Springer, 2003.
[14] H.M. Sheini and K.A. Sakallah, "From Propositional Satisfiability to Satisfiability Modulo Theories," *Theory and Applications of Satisfiability Testing—SAT '06*, pp. 1-9, Springer, 2006.
[15] L. Benini and G. De Micheli, "Networks on Chips: A New Soc Paradigm," *Computer*, vol. 35, no. 1, pp. 70-78, Jan. 2002.
[16] R.P. Dick, D.L. Rhodes, and W. Wolf, "Tgff: Task Graphs for Free," *Proc. CODES/CASHE '98: Sixth Int'l Workshop Hardware/Software Codesign*, pp. 97-101, 1998.
[17] J. Xu, W. Wolf, and W. Zhang, "Double-Data-Rate, Wave-Pipelined Interconnect for Asynchronous Nocs," *IEEE Micro*, vol. 29, no. 3, pp. 20-30, May/June 2009.
[18] J. Chong, N. Satish, B. Catanzaro, K. Ravindran, and K. Keutzer, "Efficient Parallelization of H.264 Decoding with Macro Block Level Scheduling," *Proc. IEEE Int'l Conf. Multimedia and Expo*, pp. 1874-1877, July 2007.

Weichen Liu received the BS and MS degrees from the Department of Computer Science and Engineering, Harbin Institute of Technology in 2004 and 2006, respectively. He is currently working toward PhD in the Department of Computer Science and Engineering, Hong Kong University of Science and Technology. His research interests include real-time embedded systems, real-time scheduling, software synthesis, multiprocessor systems, networks-on-chip, formal methods, and design space exploration techniques. He is a student member of the IEEE.

Zonghua Gu received the BSc degree from the University of Science and Technology of China in 1993, and the PhD degree in computer science and engineering from the University of Michigan in 2004. He is currently an associate professor in the Department of Computer Science at Zhejiang University, China. His research area is real-time embedded systems. He is a member of the IEEE.

Jiang Xu received the MA and PhD degrees in electrical engineering from Princeton University, and received the BS and MS degrees in electrical engineering from Harbin Institute of Technology. His research areas include multiprocessor system-on-chip, computer architecture, low-power VLSI design, and HW/SW codesign. From 2001 to 2002, he worked at Bell Labs as a research associate. He was a research associate at NEC Laboratories America from 2003 to 2005 and working on networks-on-chip. He joined a startup company, Sandbridge Technologies, in 2005. Since 2007, he is working in the Department of Electronic and Computer Engineering in Hong Kong University of Science and Technology as an assistant professor, and established the Mobile Computing System Lab. He has published more than 30 papers in peer-reviewed journals and conferences, and received one Best Paper Award. He serves on the organizing and technical committees in many international conferences, including ICCD, CASES, ISVLSI, VLSI, EMSOFT, VLSI-SoC, ICESS, RTCSA, NOCS, ESO, etc. He currently serves as an associate editor of *ACM Transactions on Embedded Computing Systems*. He is a member of the IEEE.

Xiaowen Wu received the BSc degree in computer science from Harbin Institute of Technology, China in 2008. He is currently working toward the PhD degree in electronic and computer engineering at the Hong Kong University of Science and Technology. His research interests include embedded systems, multiprocessor systems, Network-on-Chip, and optical Network-on-Chip. He is a student member of the IEEE.



Yaoyao Ye received the BS degree in electronic engineering from the University of Science and Technology of China, Hefei in 2008. She is currently working toward the PhD degree in electronic and computer engineering at Hong Kong University of Science and Technology. Her research focuses on optical networks-on-chip for multiprocessor systems-on-chip. She is a student member of the IEEE.